



Ligando o frontend ao backend.

Ao longo dos módulos anteriores, precisamente os que tratavam sobre o frontend, tivemos a oportunidade de conhecer e codificar utilizando a biblioteca React Js. Com o uso dessa biblioteca, é necessária a transpilação do código, processo no qual o código JSX será transformado em código HTML, CSS e Javascript “normais”. Como toda linguagem voltada para o frontend, e aqui falo especificamente do Javascript, que no trio citado é a linguagem de programação por trás do React, não é possível acessarmos os recursos de bancos de dados sem passarmos por um intermediário – o backend. Aqui, faço uma pequena ressalva: é possível utilizarmos Javascript também no backend. Caso ainda não conheça, leia sobre a linguagem Node.js .

Na estrutura do projeto template que estamos construindo ao longo dessa disciplina, temos a seguinte arquitetura de tecnologias:



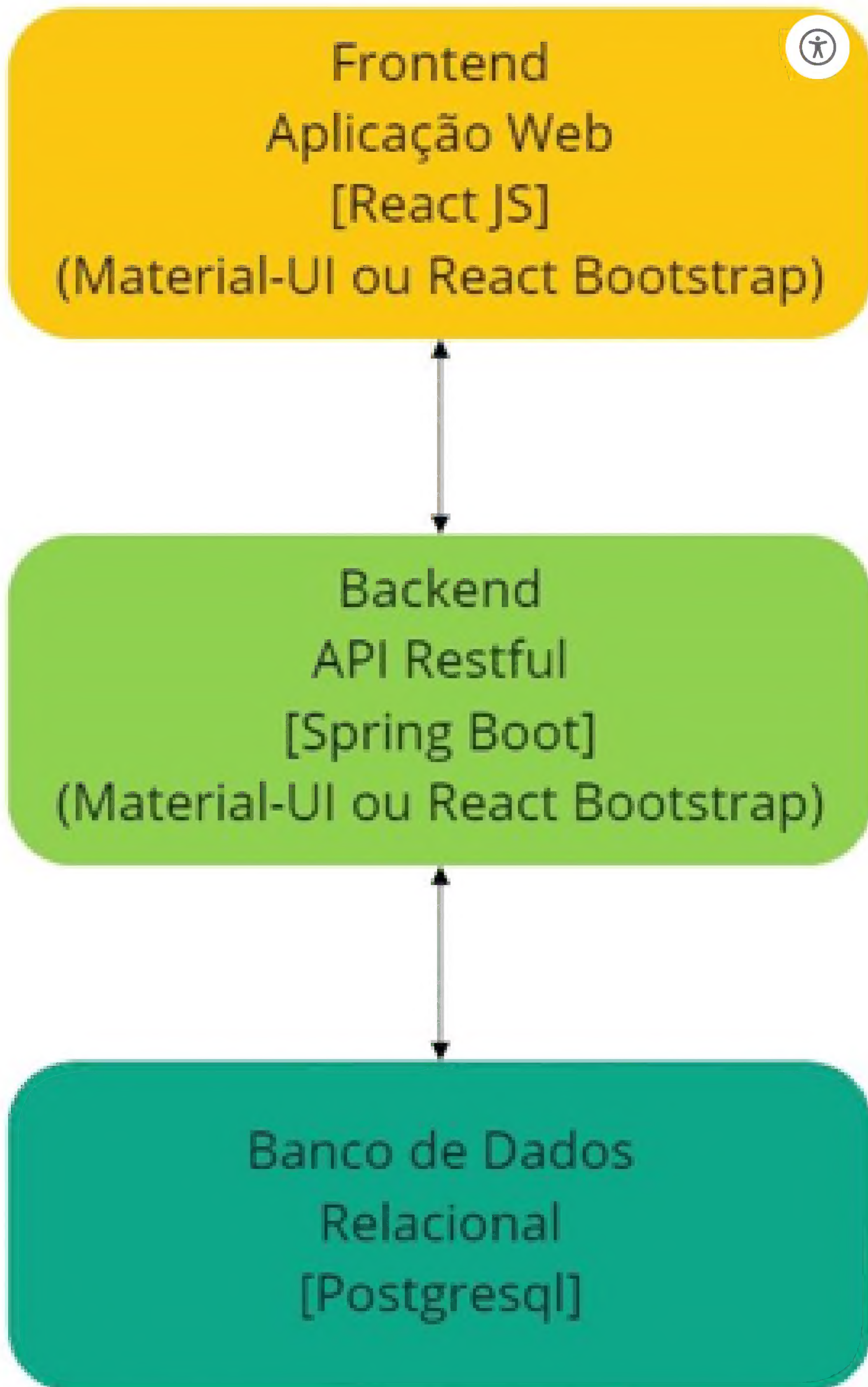


Figura 1: Arquitetura do Projeto Template



A comunicação entre cada camada na arquitetura acima é realizada por componentes específicos de cada tecnologia empregada, por exemplo, no backend, onde o framework Spring Boot é utilizado e a comunicação com o banco de dados é realizada com a api JPA. Em relação à sua comunicação com o frontend, o backend, uma API Restful, disponibiliza alguns endpoints que deverão ser acessados através do protocolo HTTP, com a troca de informações no formato JSON. Por fim, no front, para realização dessa comunicação via HTTP com o backend, precisaremos usar uma biblioteca específica, que nos permita enviar requisições HTTP e tratar suas respostas, incluindo o status das mesmas. Nesse sentido, há algumas bibliotecas e ferramentas disponíveis. Hoje, abordaremos a Axios, uma das bibliotecas mais utilizadas para a comunicação frontend(React) x backend.

A Axios ou O Axios, é um cliente HTTP baseado em promises, cuja principal tarefa é permitir a realização de requisições HTTP, tanto em aplicações/linguagens que rodam no frontend, como também no back (é compatível com o Node.js, por ex.). Entre seus principais recursos, temos:

- Permite a realização de chamadas Ajax (XMLHttpRequest);
- Tem suporte a promessas (promises);
- Permite a interceptação tanto das requisições quanto das respostas;
- Permite que requisições sejam canceladas;

- Possui conversão automática dos dados para o formato JSON;



- Entre outros.

Em relação à sua instalação, o processo é semelhante aos demais vistos até aqui:

```
npm install axios
```

OBS: o yarn também pode ser utilizado.

Normalmente, numa aplicação, é comum realizarmos requisições para diferentes endpoints, inclusive para diferentes APIs. Nesse caso, podemos facilitar nosso trabalho e evitarmos ficar declarando o endereço de cada API em cada requisição com o Axios. Para isso, criamos uma (ou mais) instância do Axios e a importamos sempre que precisarmos realizar nossas requisições. O fragmento de código abaixo demonstra como criar essa instância:

```
const instancia = axios.create({
  baseURL: 'https://endereco-da-api.com/contexto-da-api/'
});
```

Com a instância criada, podemos importá-la e utilizá-la da mesma forma que utilizamos outros componentes em React. Outro a[illegible]ue é que podemos criar diferentes instâncias, de acordo com as diferentes APIs utilizadas na aplicação – devendo ser declarada uma instância por script Javascript distinto.

A partir dessa introdução sobre o Axios, já podemos utilizá-lo. Para isso, sugiro que você crie um novo projeto – sim, repetir esse processo é ótimo para fixar os conceitos e praticarmos sempre mais – e instale as devidas dependências. Vamos utilizar uma API pública, que fornece dados fake. Na primeira tela, vamos criar uma listagem de produtos. Para isso, utilizaremos um componente da Material-UI, chamado Datagrid (você verá que se trata de um componente muito poderoso e ao mesmo tempo, muito simples de se configurar). Como Atividade Prática desse módulo, você criará, também, uma tela de cadastro de produto. Porém, vamos por partes. Comece criando um novo projeto e instalando as dependências, conforme abaixo:

```
npx create-react-app app-produto
cd app-produto
npm install axios
npm install @mui/material @emotion/react @emotion/styled @mui/x-data-grid
```

Na pasta src, crie 3 pastas:

- pages – vai armazenar as páginas da aplicação. Dentro dela, crie uma nova pasta chamada Produto e nesta pasta crie dois componentes: ListaProduto.js e CadastroProduto.js;

- components – pasta que armazenará os componentes em comum, usados nas páginas. Crie (ou copie da aplicação que vimos anteriormente) o componente header.js;

- api – pasta onde ficarão a(s) instância(s) do Axios. Como nossa aplicação só chamará uma API, crie aqui dentro o componente InstanciaAxios.js;

No ponto de entrada de nossa aplicação, ou seja, no script App.js que também fica na pasta src, lembre-se de importar o componente <Header> , seguido do componente <ListaProduto>. Logo a seguir, eu compartilharei com você o código funcional dessa aplicação, tente codificá-lo sozinho antes. Para isso, seguem algumas dicas e alguns fragmentos de código que você precisará.

A API que utilizaremos nessa aplicação é a Fakestoreapi (disponível em: https://fakestoreapi.com) **(Acesso em 30/09/2022)**. Ao acessar esse link, você encontrará toda a documentação da API, incluindo exemplos de requisição e retorno. Para a listagem de produtos, você precisará realizar um GET usando o Axios. A sintaxe é a seguinte:

```
Axios.get(
    `/endpoint-da-api`
  )
  .then(result => {
    console.log('result data: ' + JSON.stringify(result.data));
    //nesse ponto, tendo sucesso na requisição, você terá acesso ao seu retorno a partir do objeto result.data
  });
```

OBS: leia o comentário inserido acima, no fragmento de código. Além disso, lembre-se de criar a instância do Axios e importá-la no

componente de Listagem. Ao fazê-lo, você deverá trocar a linha "Axios.get" para "NomeDaSuaInstanciaAxios.get".

Em relação ao componente Datagrid, da biblioteca Material-UI, sua documentação e exemplos de uso podem ser encontrados nesse link: https://mui.com/pt/material-ui/react-table/ **(Acesso em 30/09/2022).** Em linhas gerais, você precisará:

- definir uma constante para as colunas do Datagrid;
- um state para armazenar o retorno da API;
- inserir o componente DataGrid e definir seus parâmetros, utilizando os itens acima.

Após codificar a sua aplicação, veja se ela ficou parecida com esse exemplo que eu codifiquei:

Link: https://github.com/FaculdadeDescomplica/pratica_integradora_tecnologias_disruptivas/tree/main/modulo12/app-produto **(Acesso em 29/09/2022)**

Por fim, e antes de encerrarmos este módulo, vamos dar uma olhada em alguns fragmentos de código contendo requisições feitas com o Axios. Anteriormente, já vimos o exemplo de GET. Vejamos agora o exemplo de POST (que você precisará para a atividade prática desse módulo):

```
axios.post('/user', {
  firstName: 'Santos',
  lastName: 'Dumont'
})
.then(function (response) {
  console.log(response);
})
.catch(function (error) {
  console.error(error);
});
```



Alguns pontos merecem atenção:

a) na requisição POST estão sendo enviados dois dados. Veja a sintaxe, que segue o formato de uma string JSON;

b) como o Axios implementa promises, podemos tratar o retorno com o uso do "then". Logo, o código dentro do "then" é executado em caso de sucesso na requisição;

c) em caso de falha na requisição, é executado o código dentro do "catch".

Os dois últimos pontos acima, apresentam uma questão que ainda não abordamos, mas que é muito importante: é importantíssimo tratarmos as possíveis exceções em nossos códigos. Nesse caso, no exemplo em questão, estamos tratando o seguinte: caso ocorra qualquer tipo de erro na execução da requisição, há um bloco de código para tratar tal erro. O tratamento pode ser apenas uma mensagem no console.log, porém também pode ser algo mais elaborado, como uma mensagem para o usuário que realizou a requisição, informando ter ocorrido um erro. Essa prática, de tratamento de exceções, pode e deve ser realizada em todo o nosso código.

Chegamos ao final de mais um módulo, no qual tratamos as integrações do frontend, em uma aplicação React, com o backend,

uma API Restful. Foi apresentado também o framework Axios de forma teórica e também prática, através da codificação de uma requisição para uma API fake.

Atividade Extra

Em todo código implementado, independente de estarmos usando uma linguagem de front ou de backend, é fundamental tratarmos as possíveis exceções do mesmo. O React utiliza a linguagem Javascript, que possui mecanismos de controle e tratamento de exceções. Se tiver mais interesse sobre o assunto, sugiro que acesse o link abaixo:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/try...catch **(Acesso em 29/09/2022)**

Aprenda mais sobre o Axios acessando sua documentação oficial, disponível em: https://axios-http.com/ptbr/ **(Acesso em 29/09/2022)**

Referência Bibliográfica

MORGAN, Joe. **How to code in React.js**. Digital Ocean, 2021.



Atividade Prática Módulo 12

Título da Prática: Realizando requisições POST.

Objetivos: Implementar a comunicação com o backend através de uma requisição POST.

Materiais, Métodos e Ferramentas: IDE VS Code.

Atividade Prática

Durante o módulo 12, vimos como realizar requisições, a partir da aplicação React, para uma API Restful. Nesse sentido, implementamos uma requisição Get, que retorna uma lista de produtos a partir da API https://fakestoreapi.com/ **(Acesso em 28/09/2022).** Nessa atividade, você utilizará a mesma API, mas realizará agora uma requisição POST, cadastrando um novo produto na mesma. Para isso, você deverá:

- Na mesma aplicação que criou no módulo 12, na página Produto, criar um novo componente, chamado CadastroProduto.js;
- Importar a instância do Axios;
- Verificar na documentação da Fakestoreapi, quais dados deverão ser enviados para o cadastro de produto;
- Criar um formulário utilizando a Material-UI (já fizemos isso em exercícios anteriores);

- Lembrar de seguir o padrão já visto anteriormente, de utilizar state para os dados que serão preenchidos no formulário e envi[illegible]s na requisição;

- Verificar na documentação, qual o retorno esperado em caso de erro na requisição;

- Tratar possíveis exceções com o "then" e o "catch", na requisição Axios;

- Incluir o componente CadastroProduto no App.js, para que seja exibido abaixo da listagem de produtos.

Por último, você também poderá consultar se a requisição foi executada com sucesso a partir do inspecionador de elementos, no navegador. Abra-o antes de realizar a requisição e navegue até a aba "Network" (ou Rede). Então, realize a requisição e verifique o log da mesma na aba em questão. Sendo realizada com sucesso, poderá ser visto o código de status HTTP 200 ao lado da mesma. Caso ocorra algum erro, também será possível visualizá-lo a partir do inspecionador.

Gabarito Atividade Prática

Para a realização dessa atividade são descritos alguns passos para orientar o aluno. Ao final de todo o processo, ele deverá ter criado um novo componente e realizado, com sucesso, uma requisição POST

para a Fakestoreapi, criando um novo produto na mesma. Considerando que o aluno poderá realizar suas próprias customizações, deverá ser analisado se ele conseguiu realizar a requisição POST. O código abaixo contém um exemplo do que é pedido nesta atividade e poderá ser utilizado pelo aluno para validar o seu próprio código e, consequentemente, a execução com sucesso da atividade.

Link:
https://github.com/FaculdadeDescomplica/pratica_integradora_tecnologias_disruptivas/tree/main/modulo11/atividade-pratica-gabarito
(Acesso em 28/09/2022).

Ir para exercício